Decision Management Community Challenge May 2020

Pay as you go: A Solution Using Microsoft Excel®
(and some other ways of doing it too!)

(Bob Moore, JETset Business Consulting, 20 May 2020)

1 Problem Statement (from the web site)

SaaS (Software as a Service) is taking over the cloud computing market. It offers pay-as-you-go pricing models that enable customers to pay only for what they use. Here is an example of pricing rules when your monthly bill is determined by the number of units (e.g. executed transactions) you actually use by tiers:

Tiers	Units	Cost
Free Tier	The first 10,000 units are free	$0 / unit
Tier 1	Up to 20,000 units per month	$0.050 / unit
Tier 2	From 20,000 to 50,000 units per month	$0.030 / unit
Tier 3	Above 50,000 units per month	$0.010 / unit

Now assume that you are a vendor of a SaaS product that needs to report the actual use of your product to the cloud provider in accordance with the above rules. For example, AWS requires each SaaS vendor to accumulate the actual use per customer and report the charged amount to the AWS metering system every hour by tiers. So, you need to create a pricing decision service that implements these rules. Your service will be called at the beginning of each hour and will receive 3 parameters:

- Number of all used units (from the very first use until the last hour including)
- Number of units used this month (the last hour including)
- Number of units used the last hour.

Your pricing service should produce the numbers of charged units for Tier 1, Tier 2, and Tier 3 for the last hour. Here are examples:

All units	This month units	Last hour units	Tier 1	Tier 2	Tier 3
2,000	2,000	160	0	0	0
10,200	10,200	350	200	0	0
25,600	8,678	1,234	1,234	0	0
1,500	1,500	1,500	0	0	0
22,000	22,000	20,500	10,000	2,000	0
32,500	32,500	10,500	0	10,500	0
55,600	20,500	700	200	500	0
85,600	25,700	1,500	0	1,500	0
120,258	60,390	2,350	0	0	2,350
120,000	120,000	120,000	10,000	30,000	70,000

We challenge you to implement this decision service with your favorite tool, test it (!), and submit your solutions with execution results

2 First Look

At first sight this looks easy, but there is a bit more challenge to it than it appears. The origin of the challenge was that a team attempting to build a 'real' system along this lines found working out the rules and getting them working correctly was quite time consuming.

I spent quite a bit of time trying to figure out why the various figures were the way they were, and eventually managed to explain most of them to myself, but I found it difficult to rationalise what rules I was coming up with.

However I've looked at problems a bit like this before while trying to figure out how much tax one pays on a given income and I've used Excel to do it, so I thought it could do no harm to throw everything into an Excel spreadsheet and experiment to see if I could find out what formulae I needed to put into cells to start getting similar outcomes to the test data.

3 Picking Apart the Inputs

One thing which became clear at once was that the inputs you are given are not actually the inputs you want. If you are going to work out how the units in the last hour are to be billed, you don't want to know the total number of units used by the account overall, and the total number of units used this month at the *end* of the last hour, but you want the values for these at the *beginning* of the hour.

The second thing which became evident was that having free units was a bit of a pain. But where would be the interest if there were not a few wrinkles to work out. They cause an issue because they are both 'free' but also contribute to the Tier 1 allocation.

My own idea of 'free' would be that they are just free. But actually they are better than just free, because when I start, I get to the cheaper Tier 2 level quicker. If the units were just free in my sense, of the first 30,000 units you use, 10,000 would be free, and 20,000 would be at tier 1. Looking at the examples though we see if that what actually happens in this case is you get 10,000 free, 10,000 at Tier 1 and then 10,000 at the cheaper Tier 2. This is good for the customer but arguably it makes getting the rules right a bit harder.

My third conclusion was that the only reason the total number of units used was useful, was that it was key to unravelling the problems posed by those pesky free units. More specifically if the total number of units used ***before the start*** of the last hour is more than 10,000 there are no free units left, if it is less than 10,000, the difference is the number of free units remaining at the beginning of the last hour.

So stage one of the analysis gives us three extra columns of data:

all units at end of hour	units this month at end of hour	units used during hour	all units at start of hour	units this month at start of hour	free units at start of hour
Input 1	Input 2	Input 3	Input 1 - Input 3	Input 1 – Input 2	Max (10000 - Input 1, 0)
2,000	2,000	160	1,840	1,840	8160
10,200	10,200	350	9,850	9,850	150
25,600	8,678	1,234	24,366	7,444	0
1,500	1,500	1,500	0	0	10000
22,000	22,000	20,500	1,500	1,500	8500
32,500	32,500	10,500	22,000	22,000	0
55,600	20,500	700	54,900	19,800	0
85,600	25,700	1,500	84,100	24,200	0
120,258	60,390	2,350	117,908	58,040	0
120,000	120,000	120,000	0	0	10000

This has organised the data into something we can work with more effectively. The rest of the logic now no longer needs the values in the first and fourth columns. Strictly we only need two of the second, third and fifth columns, since knowing two the third value can be derived, but it proves easier to describe the rules if we keep all three.

4 Deriving the Reporting Rules from the Billing Rules

Having distilled the data, the next step is to work out the rules which determine what numbers go in the hourly report. Having had a bit of success putting formulae into cells on my Excel spread sheet, I could see for each tier we need to consider three scenarios.

- The units used this month at the end of the hour is below the tier threshold[1]
- The units used this month crosses the tier threshold during the hour
- The units used this month at the start of the hour is above the tier threshold

In stating the scenarios in this manner, we leave the door open for adding more tiers in the future. For Tier 1 we can see the first scenario only happens if we've not used any

[1] To be clear here the tier threshold is the number of units used before it applies, so it is 0 for Tier 1, 20,000 for Tier 2 and 50,000 for Tier 3 in this case

units at all during the hour. It's a bit unclear from the problem description if we need to report anything in this scenario (or indeed if we get any data from AWS). Tier 1 also differs because there might be free units affecting the Tier 1 count, but free units do not affect the other tiers.[2]

The next step in working out the rules is to recognise that in the first scenario, no units are charged at the current tier or any higher tier. In the last scenario, no units are charged at any lower tiers, and some are charged at the current tier. But we also need to allow for the possibility that some may also be charged at a higher tier if the hourly usage is such that it crosses a higher-level tier threshold as well as the one currently of interest. The second scenario is the most complex. But after a bit of thought we can see all we need to do is subtract the difference between the tier threshold and the number of units used at the start of the hour from the number of units used during the hour. Then we can apply the logic we used in the last scenario to the remaining units[3].

In each scenario we are considering one tier and its threshold, but in the second and third scenarios units needing to be reported in higher tiers might be involved. This tells us we have to work 'backwards' determining how many units to report in the highest Tier first. Specifically we work out how many Tier 3 units to report first, then how many Tier 2 units and finally how many Tier 1 to report[4].

5 The reporting rules

On the basis of the above analysis, we can derive the following eight reporting rules:

5.1 Rules for reporting Tier 3:

We have three rules for Tier 3 following the three scenarios above:

RULE_3_1
 # if the customer still hasn't used 50,000 units for the month by the end of the last hour there are no units to report in Tier 3
If *units this month at end of hour <= 50,000*
Then *no units are reported in Tier 3*

RULE_3_2
 # if the customer passes 50,000 units for the month during this hour, everything over 50,000 units at the end of the hour is reported in Tier 3
If *units this month at start of hour <= 50,000 and units this month at end of hour > 50,000*
Then *units this month at end of hour – 50,000 units are reported in Tier 3*

RULE_3_3
 # if the customer has already used 50,000 units this month all the units are reported in Tier 3

---

[2] This is because the number of free units 10,000 is below the Tier 2 threshold 20,000. I've not looked into the complications of getting so many free units it exceeded the Tier 2 threshold!

[3] This may sound a bit convoluted, but it hopefully all will be clear when we look at the rules!

[4] Note that since the Tier 1 rules depend on data generated by the Tier 2 rules, and the Tier 2 rules on data generated in the Tier 3 rules, the order of rule execution is important. Excel conveniently manages such dependencies automatically of course

If *units this month at start of hour > 50,000*
Then *all units in hour are reported in Tier 3*

5.2 Rules for reporting Tier 2:

The logic for Tier 2 follows is quite similar, giving us three more rules, but we need to make sure we also subtract off the units which are in Tier 3, otherwise they would get double counted, so the rules will look like this:

RULE_2_1
 # if the customer still hasn't used 20,000 units for the month by the end of the last hour there are no units to report in Tier 2 (or Tier 3 either)
If *units this month at end of hour <= 20,000*
Then *no units are reported in Tier 2*

RULE_2_2
 # if the customer passes 20,000 units for the month during this hour, everything over 20,000 units at the end of the hour will be reported in Tier 2 that has not already been reported in Tier 3
If *units this month at start of hour <= 20,000 and units this month at end of hour > 20,000*
Then *(units this month at end of hour – 20,000 units – units in Tier 3) units are reported in Tier 2*

RULE_2_3
 # if the customer has already used 20,000 units this month everything not already reported in Tier 3 is reported in Tier 2
If *units this month at start of hour > 20,000*
Then *(Units used in hour – units in Tier 3) are reported in Tier 2*

5.3 Rules for reporting Tier 1:

The logic for Tier 1 is a bit different, partly because it is the lowest tier, and partly because of the effect of free units. There is no rule corresponding to the first scenario as you cannot have less than 0 units, so we only need two rules here.

RULE_1_1
 # if the customer still has free units at the end of the hour there are no billable units
If *Remaining Free Units > Units used in hour – Units in Tier 2 – Units in Tier 3*
Then *there are no units in Tier 1*

RULE_1_2
 # if the customer has no free units at the end of the hour any units not already billed in Tier 2 and Tier 3 are billed in Tier 1
If *Remaining Free Units <= Units used in hour – Units in Tier 2 – Units in Tier 3*
Then *(Units used in hour – Units in Tier 2 – Units in Tier 3 - Remaining Free Units) units are in Tier 1*

With a little thought one sees the condition in the first of these rules could be written as:

Units used in hour <= Remaining Free Units

There cannot be any units in Tier 2 or Tier 3 in this situation, since when it occurs, the total number of units used is less than 10,000 at the end of the hour.

It is also worth noting that the rules are exclusive and exhaustive. For any input, exactly one Tier 1 rule, one Tier 2 rule and one Tier 3 rule applies.

6 Implementing the Rules in Excel

Excel would not be my first choice for implementing rules, but “if it ain’t broke” as they say. So this is the implementation of the rules in Excel (I’ve used a screen shot so the column and row headers are visible):

	A	B	C	D	E	F	G	H	I
1				**Free Units**	10,000				
2		**Thresholds**		**Tier 2**	20,000				
3				**Tier 3**	50,000				
4									
5	Input Data			Derived Data From Input			Units to Report From Rules		
6	all units to end hour	units in month to end hour	units in last hour	all units to start hour	units in month to start hour	free units remaining	Tier-1	Tier-2	Tier-3
7	2,000	2,000	160	1,840	1,840	8,160	0	0	0
8	10,200	10,200	350	9,850	9,850	150	200	0	0
9	25,600	8,678	1,234	24,366	7,444	0	1,234	0	0
10	1,500	1,500	1,500	0	0	10,000	0	0	0
11	22,000	22,000	20,500	1,500	1,500	8,500	10,000	2,000	0
12	32,500	32,500	10,500	22,000	22,000	0	0	10,500	0
13	55,600	20,500	700	54,900	19,800	0	200	500	0
14	85,600	25,700	1,500	84,100	24,200	0	0	1,500	0
15	120,258	60,390	2,350	117,908	58,040	0	0	0	2,350
16	120,000	120,000	120,000	0	0	10,000	10,000	30,000	70,000

Of course the real details are hidden in the formulae of the cells. The formulae on each row are identical except for the row number, and the columns A to C are just the input data, so we only need to see what lurks behind the six cells in columns D to I on row 7. Note that the threshold parameters have been assigned ‘names’ so the formulae are easier to read and also so the tier thresholds and number of free units can easily be changed.

Column	Formula	Comments
D	=A7 - C7	Total units used at start of last hour
E	=B7 - C7	Units used this month at start of last hour
F	=MAX(Free_Units-D7, 0)	Number of free units left at start of hour
G	=IF(C7 - I7 - H7 < F7, 0, C7 - I7 - H7 - F7)	Corresponds to rules RULE_1_1 and RULE_1_2
H	=IF(E7 > Tier_2, C7 - I7, IF(B7 > Tier_2, B7 - I7 - Tier_2, 0))	Corresponds to rules RULE_2_3, RULE_2_2 and Rule_2_1 (in that order)

I	=IF(E7 > Tier_3, C7, IF(B7 > Tier_3, B7 -Tier_3, 0))	Corresponds to rules RULE_3_3, RULE_3_2 and Rule_3_1(in that order)

The cell G basically is an ***If, Then, Else*** construct while those in H and I are ***If, Then, ElseIf, Then, Else*** constructs.

7 Testing the Rules

For testing purposes one wants to ensure all possible combinations of the eight rules are exercised and demonstrated to work correctly. There are in principle 3 x 3 x 2 = 18 combinations of rule firings, but in practise there are only 7, because the conditions of firing some rules preclude the firing of others. However in 6 of these 7 cases, Rule 1_2 applies, so one may want to consider scenarios where there are free units remaining at the start of the hour and when there are not. This is possible in another 3 cases giving a total of 10 test cases to set up. The following table below summarises the combinations of rule firing which should be tested with some examples which should exercise them.

In the original examples, there are 'duplicates' where more than one example leads to the same rules firing, so I have used 6 of these which exercise distinct combinations and supplemented them with 4 new ones for full coverage.

	T1 rule	T2 rule	T 3 rule	Ok?	Example/Reason combination is impossible
1	1_1	2_1	3_1	yes	2000/2000/160 (ex 1)
2	1_1	2_1	3_2	no	3.2 cannot fire if # units < 50000 at end (implied by 1.1)
3	1_1	2_1	3_3	no	3.3 cannot fire if # units < 50000 at start (implied by 1.1)
4	1_1	2_2	3_1	no	2.2 cannot fire if # units < 20000 at end (implied by 1.1)
5	1_1	2_2	3_2	no	2.2 cannot fire if # units < 20000 at end (implied by 1.1)
6	1_1	2_2	3_3	no	2.2 cannot fire if # units < 20000 at end (implied by 1.1)
7	1_1	2_3	3_1	no	2.3 cannot fire if # units < 20000 at start (implied by 1.1)
8	1_1	2_3	3_2	no	2.3 cannot fire if # units < 20000 at start (implied by 1.1)
9	1_1	2_3	3_3	no	2.3 cannot fire if # units < 20000 at start (implied by 1.1)
10	1_2	2_1	3_1	yes	10200/10200/350 (ex 2) (free units remaining) & 10200/10200/150 (new) (no free units remaining)
11	1_2	2_1	3_2	no	3.2 cannot fire if # units < 50000 at end (implied by 2.1)
12	1_2	2_1	3_3	no	3.3 cannot fire if # units < 50000 at start (implied by 2.1)
13	1_2	2_2	3_1	yes	22000/22000/20500 (ex 5) (free units remaining) & 22000/22000/10500 (new) (no free units remaining)
14	1_2	2_2	3_2	yes	120000/120000/120000 (ex 10) (free units remaining) & 120000/120000/100000 (new) (no free units remaining)
15	1_2	2_2	3_3	no	3.3 cannot fire if # units < 50000 at start (implied by 2.2)
16	1_2	2_3	3_1	yes	32500/32500/10500 (ex 6) (no free units remaining implied by 2.3 firing)
17	1_2	2_3	3_2	yes	65000/55000/30000 (new) (no free units remaining implied by 2.3 firing)
18	1_2	2_3	3_3	yes	120258/60390/2350 (ex 9) (no free units remaining implied by 2.3 firing)

Below is the result of plugging these test cases into the Excel spreadsheet with the formulae described:

Input Data			Derived Data From Input			Units to Report From Rules		
all units to end hour	units in month to end hour	units in last hour	all units to start hour	units in month to start hour	free units remaining	Tier-1	Tier-2	Tier-3
2000	2000	160	1,840	1,840	8,160	0	0	0
10200	10200	350	9,850	9,850	150	200	0	0
10200	10200	150	10,050	10,050	0	150	0	0
22000	22000	20500	1,500	1,500	8,500	10,000	2,000	0
22000	22000	10500	11,500	11,500	0	8,500	2,000	0
120000	120000	120000	0	0	10,000	10,000	30,000	70,000
120000	120000	90000	30,000	30,000	0	0	20,000	70,000
32500	32500	10500	22,000	22,000	0	0	10,500	0
65000	55000	30000	35,000	25,000	0	0	25,000	5,000
120258	60390	2350	117,908	58,040	0	0	0	2,350

Cells in yellow are from the original examples those in cyan the new cases. I've coloured the intermediate results green, to highlight they are neither input nor output.

8 Improving with something a bit more elegant than Excel

Using Excel lead to me elucidating the rules in play, but as in implementation technology it evidently has drawbacks, apart from potential licensing issues from Microsoft, its probably not the way to communicate with AWS. It's also not ideal for testing if you are getting your coverage is really happening. If all the rules for each tier are squeezed into a single formula. How do I know which rule is actually active when I see the output?

On the other hand once one has the formulation of the rules as expressed in section 5, it's a purely mechanical process to translate the textual description of the rules into something which can be consumed by a rule engine, or even as procedural code in a language like Java, C++ or Python[5]. Some infrastructure code is needed, but as regards the rules it is literally only 16 lines of code, two lines for each rule, one for the condition, one for the consequence. An implementation in any of these technologies is then easily wrapped into a web service which one can call with the requisite inputs and get back the outputs.

Going back to why I chose Excel in the first place, my choice was influenced by having had to calculate how much tax someone needs to pay. I've done this for myself, but also for a client, as part of a mortgage affordability calculation. The client had a 'reference' implementation in Excel, but the production system used a rule engine[6].

[5] Actually I cut and pasted the rules from section five, and after a number of judicious global substitutions turned it into a Python implementation of the rules. Then by adding a bit of logging I could verify that the test cases really did met the coverage criteria. This code was NOT the code shown in section 9

[6] Just to put this into context we are talking here about less than twenty rules in an overall decision service with a thousand or more

From a maintenance perspective one pain in doing this is that the tax bands in the UK change pretty much every year. And every so often new tax bands appear, or occasionally disappear. The bad aspects of both the Excel reference code and the rules in the rule base were that in the client's existing implementation, thresholds for each band, and the number of bands were 'hard coded' into the Excel cell formulae and into the rules themselves. Maintenance of both was decidedly messy as the thresholds were explicitly mentioned in multiple places both within single formulae and rules and across them as well. Updating was time consuming and because changes were needed in so many places, extensive testing needed to ensure no mistakes had been made.

Using 'names' (as in the Excel solution above) isolates changes in the thresholds (one does not need to change any formulae to alter the number of free units, or when Tiers 2 and 3 kick in), but to add in a new tier would not only need a new column but updates to all the existing formulae as well. When on my mortgage project, naturally I wasn't allowed to touch the reference implementation, but I did find a quite neat way to improve the run time implementation of the tax band calculation rules, storing the thresholds in exactly one place and having the rules indifferent to the number of tax bands. Can the same thing be done here to allow for an arbitrary number of tiers?

The idea is simply to recognise that the rules (at least in tier 2 and 3) are really the same except in that tier 2, we have a different threshold and we need to account for units already counted in tier 3. Likewise in tier 1 the Rule 1_2, considers units already counted in tier 2 and tier 3. So were we to add a tier 4, we'd end up with very similar rules, but with lower level tiers having to take into account units counted in tier 4 as well.

Rather than write separate rules with separate tier thresholds and separately counting the units allocated to the other tiers, why not write rules which take the threshold and the accumulated allocation as input parameters? In that case you only need three rules for all the tiers, not three for each.

Once you've got the idea it is very simple to actually implement. And in fact what you end up with is simpler all round. The logic *can* be implemented in only 14 lines of Python[7] – 2 less than I suggested are needed to implement the 8 rules described in section 5. And effectively you end up with only two 'rules'[8]. Below is the code but bulked out considerably[9] with comments to explain things, and to provide some harness code to run it from the command line. It handles the existing tiers but shows how to add more if you wish or indeed remove them!

[7] Two functions one six lines long, one eight. But this does leave out useful things like comments! It would take a bit more code in something like Java or C++, but not a lot, nor in a rule engine like Blaze Advisor

[8] It is a bit disingenuous, Rule 1_1 about all the units coming out of the free allowance is managed in a slightly different manner, and rules like Rule 2_1 and 3_1 are avoided by initialising data to zero,

[9] It's 93 lines not 14!

9 A procedural solution using Python and supporting multiple tiers

```
# processTier - generate the billable units in a given tier. It assumes that higher level tiers
# (if any) have already been processed
# Returns: no value is returned, the effect of the function is to update tierCounts[tierIdx]
# Parameters are:
# tierCounts         - an array holding the billable units in each tier these are initialised
#                      to zero and updated working from higher level tiers to lower level ones
# thisMonthAtEnd     - units used at end of month (from input)
# lastHour           - units used in last hour
# tierIdx            - index of tier we are getting billable units for (zero based index)
# tiers              - the tier thresholds as an array of values above which tiers apply
# remainingFreeUnits - the number of remaining free units
def processTier(tierCounts, thisMonthAtEnd, lastHour, tierIdx, tiers, remainingFreeUnits):
    # we calculate how many units are already 'allocated'
    # Sum up all the entries in 'higher level' tiers for which Python has a snazzy syntax
    # If this is the lowest tier (index 0) also take account of any remaining free units
    allocated = sum(tierCounts[tierIdx:]) + (remainingFreeUnits if tierIdx == 0 else 0)

    # There is no if statement equivalent to RULE_2_1 or RULE_3_1 in the text
    # because the tierCount is initialised to zero

    # This if statement corresponds to RULE_2_2 and RULE_3_2 in the text but for any number of tiers
    if thisMonthAtEnd - lastHour <= tiers[tierIdx] and thisMonthAtEnd > tiers[tierIdx]:
        tierCounts[tierIdx] = thisMonthAtEnd - tiers[tierIdx] - allocated

    # This if statement corresponds to RULE_1_2, RULE_2_3 and RULE_3_3 in the text but for any number of tiers
    if thisMonthAtEnd - lastHour > tiers[tierIdx]:
        tierCounts[tierIdx] = lastHour - allocated

# generateReport - generates a report based on usage data
```

```
# Returns: an array of numbers giving the number of billable units in each tier
# Parameters are:
# total          - total number of units used so far by account since opened
# thisMonthAtEnd - number of units so far used this month
# lastHour       - number of units used in last hour
# freeUnits      - the number of free units allocated to the account on opeing
# tiers          - the tier thresholds as an array of values above which tiers apply
def generateReport(total, thisMonthAtEnd, lastHour, freeUnits, tiers):
    # how many tiers are there?
    tierNum = len(tiers)

    # tierCounts is the output - an array with the billable units to be reported
    # we create it with an entry for each tier initialised to 0 - a bit more snazzy Python syntax
    tierCounts = [0] * tierNum

    # calculate any remaining free units
    remainingFreeUnits = max(0, freeUnits - total + lastHour)

    # if there are still free units left at the end of the hour we can stop now (this is Rule 1_1!)
    if lastHour > remainingFreeUnits:
        # otherwise call processTier to calculate the number of units for each
        # tier working from the highest to the lowest
        for tierIdx in range(tierNum - 1, -1, -1):
            processTier(tierCounts, thisMonthAtEnd, lastHour, tierIdx, tiers, remainingFreeUnits)

    # and finally return the results!
    return tierCounts

# the following logic allows you to run the program from the command line
# just pass in the three input paramters, total units used, monthly units used, last hour units used
# and it prints out the report information
# or altenatively give the name of a tab delimited file with a list of inputs for bulk cases
```

```
import sys

if __name__ == '__main__':
    freeUnits = 10000 # number of free units on account opening
    tiers = [0, 20000, 50000] # default tiers from challenge
    #tiers = [0, 20000, 35000, 50000] # uncomment this line for one extra tier
    #tiers = [0, 20000, 30000, 40000, 50000] # uncomment this line for two extra tiers
    #tiers = [0, 20000] # or this one for only two tiers

    if len(sys.argv) > 3:
        tierCounts = generateReport(int(sys.argv[1]), int(sys.argv[2]), int(sys.argv[3]), freeUnits, tiers)
        report = ''
        for tierIdx, tierCount in enumerate(tierCounts):
            print("Tier %d has %6d units billable "%(tierIdx+1, tierCount))
    elif len(sys.argv) > 1:
        inFile = open(sys.argv[1], 'r')
        for line in inFile:
            (total, thisMonth, lastHour) = line.split('\t')
            tierCounts = generateReport(int(total), int(thisMonth), int(lastHour), freeUnits, tiers)
            outline = ''
            for tierCount in tierCounts:
                outline += "%d\t"%tierCount
            print(outline[:-1])
        inFile.close()
    else:
        print("Usage provide: total units used, total units used this month, units used last hour")
        print("or: a tab delimited file these three values on each line")
```